Title: Demonstrate reusability of code thru Inheritance and use of exception handling
Objectives: 1) To learn and understand code reusability and demonstrate it using Inheritance concepts.
2) To learn, understand and demonstrate exception handling in object oriented environment.
Problem Statement: Imagine a publishing company which does marketing for book and audiocassette versions. Create a class publication that stores the title (a string) and price (type float) of a publication. From this class derive two classes: book, which adds a page count (type int), and tape, which adds a playing time in minutes (type float).
Write a program that instantiates the book and tape classes, allows user to enter data and displays the data members. If an exception is caught, replace all the data member values with zero values.
Outcomes: 1) Students will be able to learn and understand inheritance and exception handling.
2) Students will be able to demonstrate inheritance and exception handling.
Hardware requirements: Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.
Software requirements: 64 bit Linux/Windows Operating System, G++ compiler
Theory:
Inheritance: C++ supports the concept of reusability once a class has been written and tested, it can be adapted by other programmers to suit heir requirements. This is basically done by creating new classes, reusing the properties of the existing ones. The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as base class or super class. And the new one is called as derived class or subclass.
Base Class & Derived Class: When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base class**, and the new class is referred to as the **derived class.** A derived class represents a more specialized group of objects. Typically, a derived class contains behaviors inherited from its base class plus additional behaviors. A class can be derived from more than one class, which means it can inherit data and functions from multiple base classes.
To define a derived class, we use a class derivation list to specify the base class(es).
A class derivation list names one or more base classes and has the form:

class derived-class : access-specifier base-class

Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default. C++ offers three forms of inheritance—public, protected and private.
Access Control and Inheritance: A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

We can summarize the different access types according to who can access them in the following way:

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance. The type of inheritance is specified by the access-specifier as explained above. We hardly use protected or private inheritance, but public inheritance is commonly used. While using different type of inheritance, following rules are applied:

1) **Public Inheritance:** When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.

2) **Protected Inheritance:** When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.

3) **Private Inheritance:** When deriving from a private base class, public and protected members of the base class become private members of the derived class.

Types of Inheritances: New classes can be built from the existing classes. It means that we can add additional features to an existing class without modifying it. The new class is referred as derived class or subclass and the original class is known as base classes or super class. Following are the types of Inheritance:

- Single Level Inheritance
- Multiple Inheritances
- Hierarchical inheritance
- Multilevel Inheritance
- Hybrid Inheritance.

Exception Handling : An exception occurs when an unexpected error or unpredictable behaviors happened on your program not caused by the operating system itself. These exceptions are handled by code which is outside the normal flow of control and it needs an emergency exit. Compared to the structured exception handling, returning an integer as an error flag is problematic when dealing with objects. The C++ exception-handling can be a full-fledged object, with data members and member functions. Such an object can provide the exception handler with more options for recovery. A clever exception object, for example, can have a member function that returns a detailed verbal description of the error, instead of letting the handler look it up in a table or a file.

C++ has incorporated three operators to help us handle these situations: try, throw and catch.

The following is the try, throw…catch program segment example:
Syntax:

```
try

{

Compound-statement handler-list

handler-list here

The throw-expression:

throw expression

}

catch (exception-declaration) compound-statement

{

Exception-declaration:

type-specifier-list here

}
```

Try: A try block is a group of C++ statements, enclosed in curly braces { }, that might cause an exception. This grouping restricts the exception handlers to the exceptions generated within the try block. Each try block may have one or more associated catch blocks.
If no exception is thrown during execution of the guarded section, the catch clauses that follow the try block are not executed or bypassed. Execution continues at the statement after the last catch clause following the try block in which the exception was thrown.
If an exception is thrown during execution of the guarded section or in any routine the guarded section calls either directly or indirectly such as functions, an exception object will be created from the object created by the throw operand.
At this point, the compiler looks for a catch clause in a higher execution context that can handle an exception of the type thrown or a catch handler that can handle any type of exception. The compound-statement after the try keyword is the guarded section of code.
Throw: The throw statement is used to throw an exception and its value to a matching catch exception handler. A regular throw consists of the keyword throw and an expression.
The result type of the expression determines which catch block receives control. Within a catch block, the current exception and value may be re-thrown simply by specifying the throw keyword alone that is without the expression.
The throw is syntactically similar to the operand of a return statement but here, it returns to the catch handler.
catch:

1) A catch block is a group of C++ statements that are used to handle a specific thrown exception. One or more catch blocks, or handlers, should be placed after each try block. A catch block is specified by the keyword catch.
2) A catch parameter, enclosed in parentheses (), which corresponds to a specific type of exception that may be thrown by the try block.
3) A group of statements, enclosed in curly braces { }, whose purpose is to handle the exception

Algorithm:
1) Create a class Marketing with variable title and price.
2) Define default and parameterized constructors to initialize title and price variables
3) Define function getData() to accept values from user
4) Define a function putData() to print these values.
5) In putData(), check if length of title is more than 3 chars else generate exception and handle exception.
6) In putData(), check if price is non negative else generate exception and handle exception.
7) Create a derived class Book from base Marketing with variable number of pages.
8) Define default and parameterized constructors to initialize variables
9) Define function getData() to accept values from user
10) Define a function putData() to print these values.
11) In putData(), check if number of pages is non negative else generate exception and handle exception.
12) Create a derived class Cassette from base Marketing with variable number of pages.
13) Define default and parameterized constructors to initialize variables
14) Define function getData() to accept values from user
15) Define a function putData() to print these values.
16) In putData(), check if number of pages is non negative else generate exception and handle exception.
17) In main function, create two objects, one of Book class and other of Cassette class. Use appropriate constructors and methods to print details of Book and Cassette.

Flowchart:
Draw proper flowchart and get it corrected from faculty.
Test Cases:
Test Case 1: (All valid inputs are given and desired output is shown)
Enter Title : C++
Enter Price : 250.34
Enter No of pages of book : 356
Enter Title : Vande Mataram
Enter Price : 156
Enter Play time in minutes : 34.54

Test Case 2: (Base class require title length more than 3 chars and price more than 0 otherwise exception will be generated)
Enter Title : C
Enter Price : -250.34
Enter No of pages of book : 356
Enter Title : Vande Mataram
Enter Price : -156
Enter Play time in minutes : 34.54
Error Msg : Title can not be less than three characters Price cann't be negative
Test Case 3: (Book require non negative number of pages and Cassette needs non negative play time)
Enter Title : C++
Enter Price : 250.34
Enter No of pages of book : -356
Enter Title : Vande Mataram
Enter Price : 156
Enter Play time in minutes : -34.54
Error Msg : Number of Pages cann't be negative
Play time must be more than 0
Conclusion: Hence, we learnt to use and demonstrate concepts of inheritance and exception handling.

PROGRAM

```
#include <iostream>
 using namespace std;
 class Marketing
 {
 public:
 Marketing()
 {
 title="";
 price=0.0;
 }
 Marketing(string title, float price)
 {
 this->title=title;
 this->price=price;
 }
 void getData()
 {
 cout<<"\nEnter title and price\n";
 cin>>title>>price;
 }
 void putData()
 {
 try
 {
 if(title.length()<3)
```

```
throw title;
if(price<=0.0)
throw price;
}
catch(string)
{
cout<<"\nError: Title below 3 characters is not allowed";
title="";
}
catch(float f)
{
cout<<"\nError: Price not valid: \t"<<f;
price=0.0;
}
cout<<"\nTitle is :"<<title;
cout<<"\nPrice is :"<<price;
}
private:
string title;
float price;
};
class Book: public Marketing
{
public:
Book():Marketing()
{
pages=0;
}
Book(string title, float price, int pages):Marketing(title,price)
{
this->pages=pages;
}
void getData()
{
Marketing::getData();
cout<<"\nEnter no. of pages in book\n";
cin>>pages;
}
void putData()
{
Marketing::putData();
try
{
if(pages<0)
throw pages;
}
catch(int f)
{
cout<<"\nError: Pages not valid: \t"<<f;
pages=0;
}
cout<<"\nPages are :"<<pages;
}
private:
int pages;
};
class Cassette: public Marketing
```

```
{
public:
Cassette():Marketing()
{
playtime=0.0;
}
Cassette(string title, float price, float playtime):Marketing(title,price)
{
this->playtime=playtime;
}
void getData()
{
Marketing::getData();
cout<<"\nEnter play time of cassette\n";
cin>>playtime;
}
void putData()
{
Marketing::putData();
try
{
if(playtime<0.0)
throw playtime;
}
catch(float f)
{
cout<<"\nError: Playtime not valid: \t"<<f;
playtime=0.0;
}
cout<<"\nPlaytime is :"<<playtime;
}
private:
float playtime;
};
int main()
{
Book book;
cout<<"\n**************BOOK*************\n";
book.getData();
cout<<"\n**************CASSETTE*************\n";
Cassette cassette;
cassette.getData();
cout<<"\n**************BOOK*************\n";
book.putData();
cout<<"\n**************CASSETTE*************\n";
cassette.putData();
return 0;
}
```

/*

**********Output **********

**********BOOK**********

Enter title and price

c++

111.44

Enter no. of pages in book
234
**********CASSETTE**********
Enter title and price
ddlj
100
Enter play time of cassette
23.4
**********BOOK**********
Title is :c++
Price is :111.44
Pages are :234
**********CASSETTE**********
Title is :ddlj
Price is :100
Playtime is :23.4
OUTPUT 2:
**********BOOK**********
Enter title and price
C
100
Enter no. of pages in book
-34
**********CASSETTE**********
Enter title and price
DDLJ
-100
Enter play time of cassette
23.4
**********BOOK**********
Error: Title below 3 characters is not allowed
Title is :
Price is :100
Error: Pages not valid: -34
Pages are :0
**********CASSETTE**********
Error: Price not valid: -100
Title is :DDLJ
Price is :0
Playtime is :23.4
*/